Class 8 Mini-Project:*

Unsupervised Learning Analysis of Human Breast Cancer Cells

Barry Grant

2022-10-10

1. Overview

The goal of this mini-project is for you to explore a complete analysis using the unsupervised learning techniques covered in class. You'll extend what you've learned by combining PCA as a preprocessing step to clustering using data that consist of measurements of cell nuclei of human breast masses. This expands on our UK food and RNA-Seq analysis from last day.

The data itself comes from the *Wisconsin Breast Cancer Diagnostic Data Set* first reported by *K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets"*.

Values in this data set describe characteristics of the cell nuclei present in digitized images of a fine needle aspiration (FNA) of a breast mass.

FNA is a type of biopsy procedure where a very thin needle is inserted into an area of abnormal tissue or cells with a guide of CT scan or ultrasound monitors (Figure 1). The collected sample is then transferred to a pathologist to study it under a microscope and examine whether cells in the biopsy are normal or not.

Features measured include `radius` (mean of distances from center to points on the perimeter), `texture` (i.e. standard deviation of gray-scale values), and `smoothness` (local variation in radius lengths). Summary information is also provided for each group of cells including `diagnosis` (i.e. benign (not cancerous) and malignant (cancerous)).

*http://thegrantlab.org/teaching/

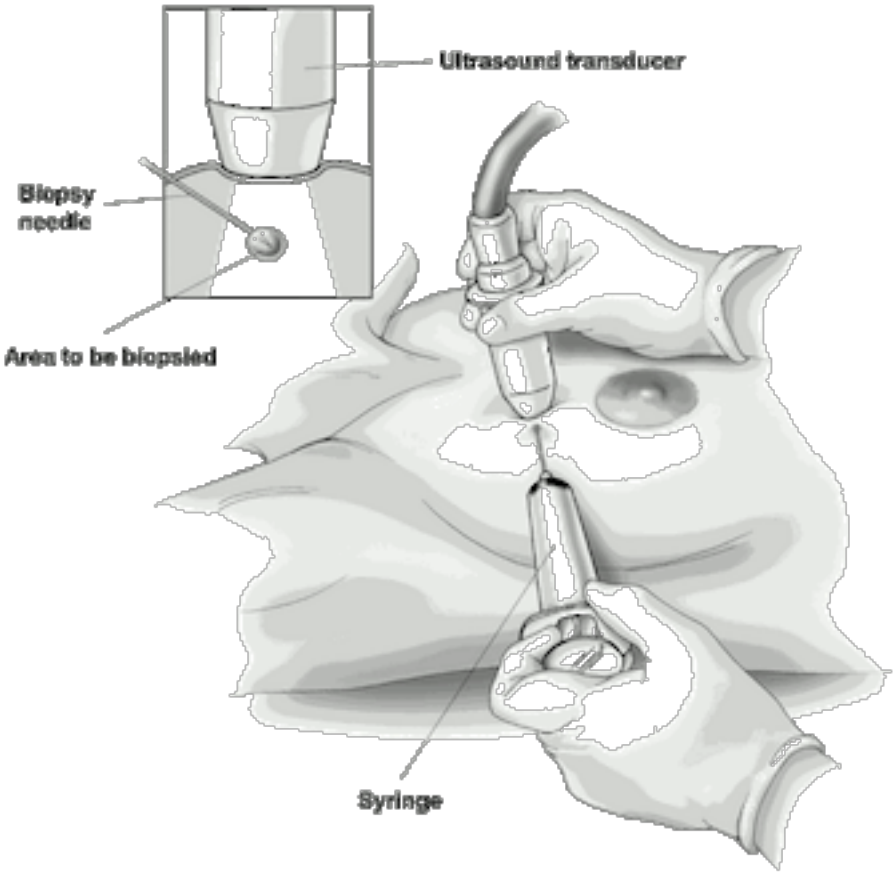


Figure 1: Fine needle aspiration (FNA) biopsy procedure.

2. Exploratory data analysis

Getting organized

Before analyzing any dataset we need to get ourselves organized by following the steps below:

- First open a new **RStudio Project** for your work called `class08` (i.e. click: `File` > `New Project` > `New Directory` > `New Project`, making sure to save this along with the rest of your course work for this class).
- Then open a new **Quarto document** (`File` > `New File` > `Quarto document`) for saving your code and accompanying narrative notes. We will use this to generate our final project report later.
- In your new Quarto document be sure to keep the YAML header but remove the boilerplate example text and code (i.e. delete from line 6 onward) so you have a clean document to work in and **add your content** to.
- Test if you can add some content and generate a PDF or HTML report (i.e. Use the **Render** button/option in RStudio) as we will need this for project submission to gradescope later.
- Note that your report should clearly indicate your answers for all **Questions**. Question 11 and 15 are optional extensions and are not required in your report.

Preparing the data

Before we can begin our analysis we first have to download and import our data correctly into our R session.

For this we can use the `read.csv()` function to read the CSV (comma-separated values) file containing the data (available from our class website: WisconsinCancer.csv)

Assign the result to an object called `wisc.df`.

```
# Save your input data file into your Project directory
fna.data <- "WisconsinCancer.csv"

# Complete the following code to input the data and store as wisc.df
wisc.df <- ___(fna.data, row.names=1)
```

Examine your input data to ensure column names are set correctly. The `id` and `diagnosis` columns will not be used for most of the following steps (you can use the **View()** or **head()** functions here).

```
head(wisc.df, 4)
```

```
         diagnosis radius_mean texture_mean perimeter_mean area_mean
842302           M       17.99        10.38         122.80    1001.0
842517           M       20.57        17.77         132.90    1326.0
84300903         M       19.69        21.25         130.00    1203.0
84348301         M       11.42        20.38          77.58     386.1
         smoothness_mean compactness_mean concavity_mean concave.points_mean
842302           0.11840          0.27760         0.3001             0.14710
842517           0.08474          0.07864         0.0869             0.07017
84300903         0.10960          0.15990         0.1974             0.12790
84348301         0.14250          0.28390         0.2414             0.10520
         symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
842302          0.2419                0.07871    1.0950     0.9053        8.589
842517          0.1812                0.05667    0.5435     0.7339        3.398
84300903        0.2069                0.05999    0.7456     0.7869        4.585
84348301        0.2597                0.09744    0.4956     1.1560        3.445
         area_se smoothness_se compactness_se concavity_se concave.points_se
842302    153.40      0.006399        0.04904      0.05373           0.01587
842517     74.08      0.005225        0.01308      0.01860           0.01340
84300903   94.03      0.006150        0.04006      0.03832           0.02058
84348301   27.23      0.009110        0.07458      0.05661           0.01867
         symmetry_se fractal_dimension_se radius_worst texture_worst
842302       0.03003             0.006193        25.38         17.33
842517       0.01389             0.003532        24.99         23.41
84300903     0.02250             0.004571        23.57         25.53
84348301     0.05963             0.009208        14.91         26.50
         perimeter_worst area_worst smoothness_worst compactness_worst
842302            184.60     2019.0           0.1622            0.6656
842517            158.80     1956.0           0.1238            0.1866
84300903          152.50     1709.0           0.1444            0.4245
84348301           98.87      567.7           0.2098            0.8663
         concavity_worst concave.points_worst symmetry_worst
842302            0.7119               0.2654         0.4601
842517            0.2416               0.1860         0.2750
84300903          0.4504               0.2430         0.3613
84348301          0.6869               0.2575         0.6638
         fractal_dimension_worst
842302                   0.11890
842517                   0.08902
84300903                 0.08758
84348301                 0.17300
```

Note that the first column here `wisc.df$diagnosis` is a pathologist provided expert diagnosis. We will not be using this for our unsupervised analysis as it is essentially the "answer" to the question which cell samples are malignant or benign.

To make sure we don't accidentally include this in our analysis, let's create a new data.frame that omits this first column:

```
# We can use -1 here to remove the first column
wisc.data <- wisc.df[,-1]
```

Finally, setup a separate new vector called `diagnosis` that contains the data from the diagnosis column of the original dataset. We will store this as a factor (useful for plotting) and use this later to check our results.

```
# Create diagnosis vector for later
diagnosis <- ___
```

Exploratory data analysis

Before diving into PCA or clustering, take a few minutes to understand the structure of your data—how many samples, how many features, and whether there are any obvious issues like missing values or unexpected column types.

Explore the data you created before (`wisc.data` and `diagnosis`) to answer the following questions:

- **Q1**. How many observations are in this dataset?
- **Q2**. How many of the observations have a malignant diagnosis?
- **Q3**. How many variables/features in the data are suffixed with `_mean`?

Tip

The functions `dim()`, `nrow()`, `table()`, `length()` and `grep()` may be useful for answering the first 3 questions above.

3. Principal Component Analysis

Performing PCA

The next step in your analysis is to perform principal component analysis (PCA) on `wisc.data`.

It is important to check if the data need to be scaled before performing PCA. Recall two common reasons for scaling data include:

- The input variables use different units of measurement.
- The input variables have significantly different variances.

Check the mean and standard deviation of the features (i.e. columns) of the `wisc.data` to determine if the data should be scaled. Use the `colMeans()` and `apply()` functions like you've done before.

```
# Check column means and standard deviations
colMeans(wisc.data)

apply(wisc.data,2,sd)
```

Execute PCA with the `prcomp()` function on the `wisc.data`, scaling if appropriate, and assign the output model to `wisc.pr`.

```
# Perform PCA on wisc.data by completing the following code
wisc.pr <- prcomp( ___ )
```

Inspect a summary of the results with the `summary()` function.

```
# Look at summary of results
summary(wisc.pr)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7
Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880 0.82172
Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025 0.02251
Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759 0.91010
```

- **Q4**. From your results, what proportion of the original variance is captured by the first principal component (PC1)?
- **Q5**. How many principal components (PCs) are required to describe at least 70% of the original variance in the data?
- **Q6**. How many principal components (PCs) are required to describe at least 90% of the original variance in the data?

Interpreting PCA results

Now you will use some visualizations to better understand your PCA model. A common visualization for PCA results is the so-called biplot.

However, you will often run into some common challenges with using biplots on real-world data containing a non-trivial number of observations and variables. Here we will need to look at some alternative visualizations. You are encouraged to experiment with additional visualizations before moving on to the next section.

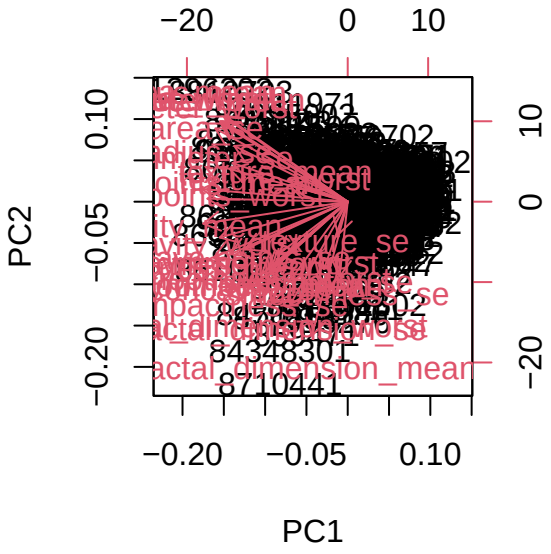
Create a biplot of the `wisc.pr` using the `biplot()` function.

- **Q7.** What stands out to you about this plot? Is it easy or difficult to understand? Why?

  Tip

  This is a hot mess of a plot and we will need to generate our own plots to make sense of this PCA result.

```
biplot(___)
```

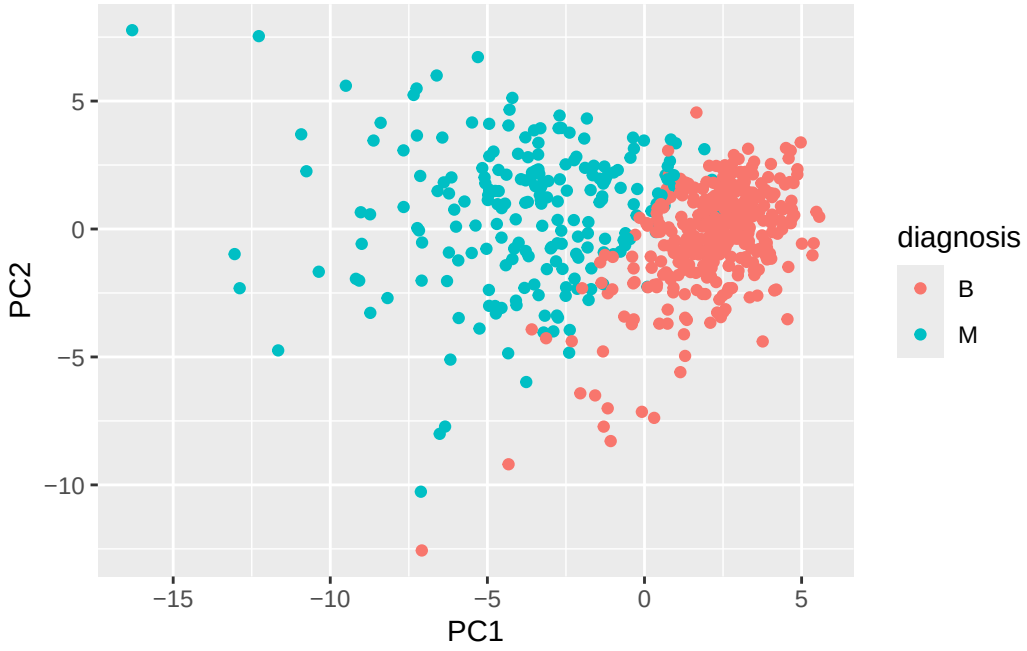


Rownames are used as the plotting character for biplots like this one which can make trends rather hard to see. In fact, this plot is very poor. So lets generate a more standard scatter plot

of each observation along principal components 1 and 2 (i.e. a plot of PC1 vs PC2 available as the first two columns of `wisc.pr$x`) and color the points by the diagnosis (available in the `diagnosis` vector you created earlier).

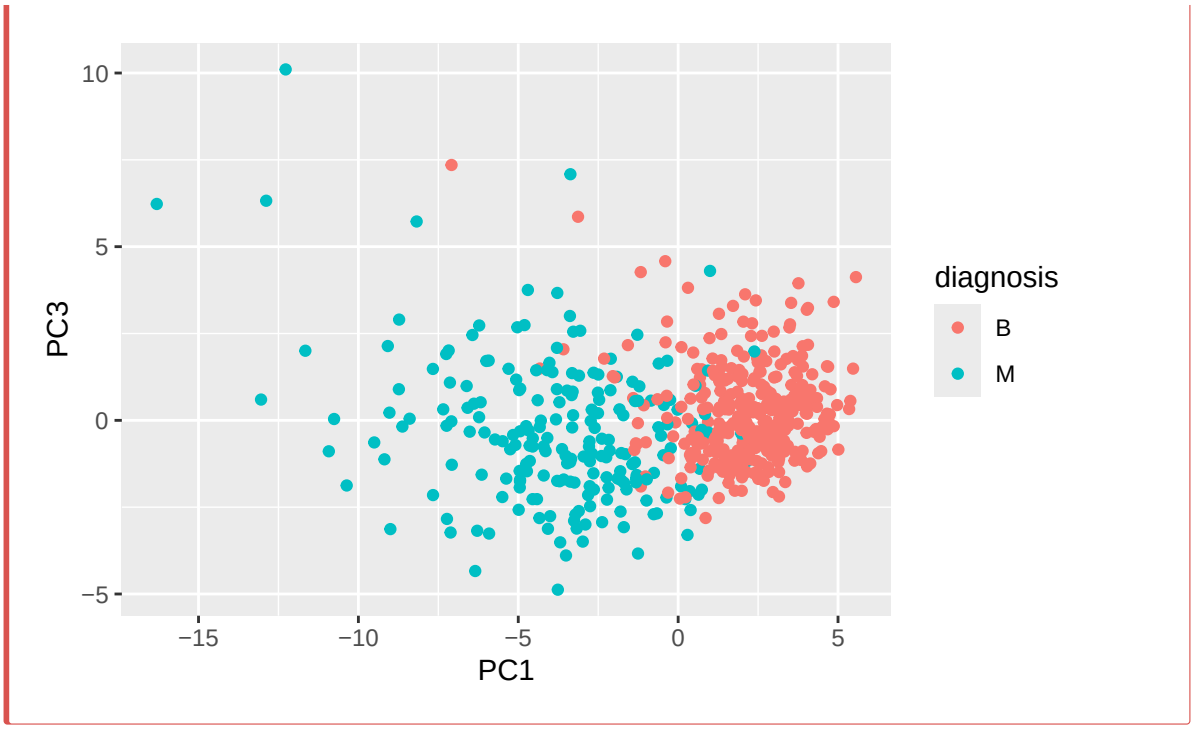
```
# Scatter plot observations by components 1 and 2
library(ggplot2)

ggplot(wisc.pr$x) +
  aes(___, ___, col=diagnosis) +
  geom____()
```



- **Q8.** Generate a similar plot for principal components 1 and 3. What do you notice about these plots?

```
# Repeat for components 1 and 3
ggplot(___) +
  aes(PC1, ___, col=diagnosis) +
  ___()
```



Overall, the plots indicate that principal component 1 is capturing a separation of malignant (turquoise) from benign (red) samples. This is an important and interesting result worthy of further exploration - as we will do in the next sections!

Variance explained

A scree plot shows how much variance each PC captures. We typically look for an "elbow" — a point where adding more PCs gives diminishing returns. This can help us decide how many PCs to consider for further analysis. (Spoiler: some real data sets don't have a perfect elbow, so folks will often use a threshold like 70% or 90% cumulative variance instead.)

Calculate the variance of each principal component by squaring the `sdev` component of `wisc.pr` (i.e. `wisc.pr$sdev^2`). Save the result as an object called `pr.var`.

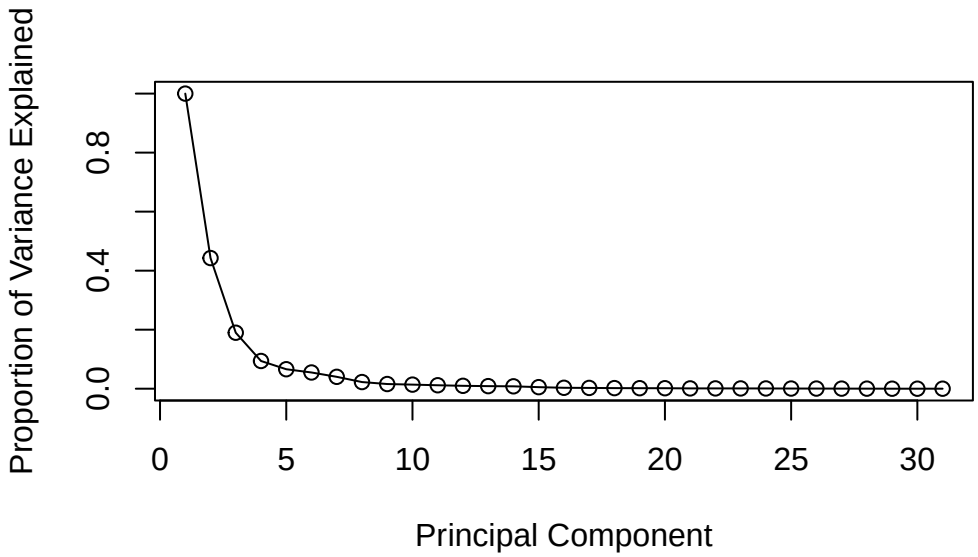
```
# Calculate variance of each component
pr.var <- ___
head(pr.var)
```

```
[1] 13.281608  5.691355  2.817949  1.980640  1.648731  1.207357
```

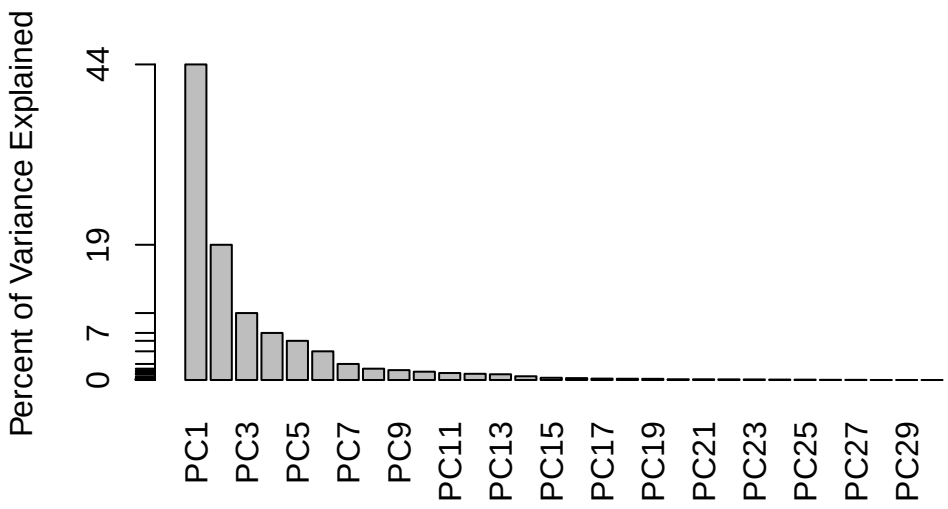
Calculate the variance explained by each principal component by dividing by the total variance explained of all principal components. Assign this to a variable called `pve` and create a plot of variance explained for each principal component.

```
# Variance explained by each principal component: pve
pve <- ___ / ___

# Plot variance explained for each principal component
plot(c(1,pve), xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0, 1), type = "o")
```

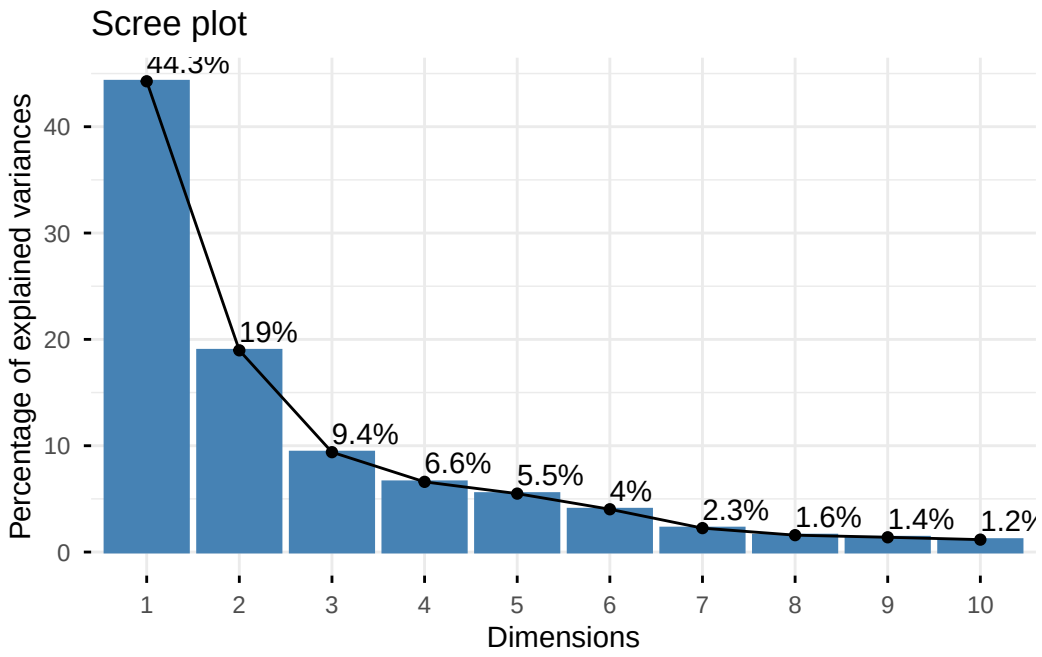



```
# Alternative scree plot of the same data, note data driven y-axis
barplot(pve, ylab = "Percent of Variance Explained",
     names.arg=paste0("PC",1:length(pve)), las=2, axes = FALSE)
axis(2, at=pve, labels=round(pve,2)*100 )
```



OPTIONAL: There are quite a few CRAN packages that are helpful for PCA. This includes the **factoextra** package. Feel free to explore this package. For example:

```
## ggplot based graph
#install.packages("factoextra")
library(factoextra)
fviz_eig(wisc.pr, addlabels = TRUE)
```



Communicating PCA results

In this section we will check your understanding of the PCA results, in particular the *"loadings"* and *"variance explained"*.

The loading vector (`wisc.pr$rotation`) tells us which original measurements contribute most to each PC.

A large PC1 loading value (positive or negative) for one of the 30 original measurements (i.e. features/columns we started with), for example, would suggest that this feature is an important driver of the variation we see in the score plot and thus helpful for distinguishing "M" from "B" samples. Let's check which features matter most to PC1.

- **Q9.** For the first principal component, what is the component of the loading vector (i.e. `wisc.pr$rotation[,1]`) for the feature `concave.points_mean`? This tells us how much this original feature contributes to the first PC. Are there any features with larger contributions than this one?

4. Hierarchical clustering

The goal of this section is to do hierarchical clustering of the **original data**to see if there is any obvious grouping into malignant and benign clusters.

Recall from class that hierarchical clustering does not assume in advance the number of natural groups that exist in the data (unlike K-means clustering).

As part of the preparation for hierarchical clustering, the distance between all pairs of observations needs to be calculated. This "distance matrix" will be the input for the `hclust()` function.

One of the optional arguments to `hclust()` allows you to pick different ways (a.k.a. "methods") to link clusters together, with `single`, `complete`, and `average` being the most common "linkage methods". You can explore the effectes of these different methods in this section.

First scale the `wisc.data` data and assign the result to `data.scaled`.

```
# Scale the wisc.data data using the "scale()" function
data.scaled <- ___(wisc.data)
```

Next, calculate the (Euclidean) distances between all pairs of observations in the new scaled dataset and assign the result to `data.dist`.

```
data.dist <- ___(data.scaled)
```

Create a hierarchical clustering model using complete linkage. Manually specify the method argument to hclust() and assign the results to `wisc.hclust`.

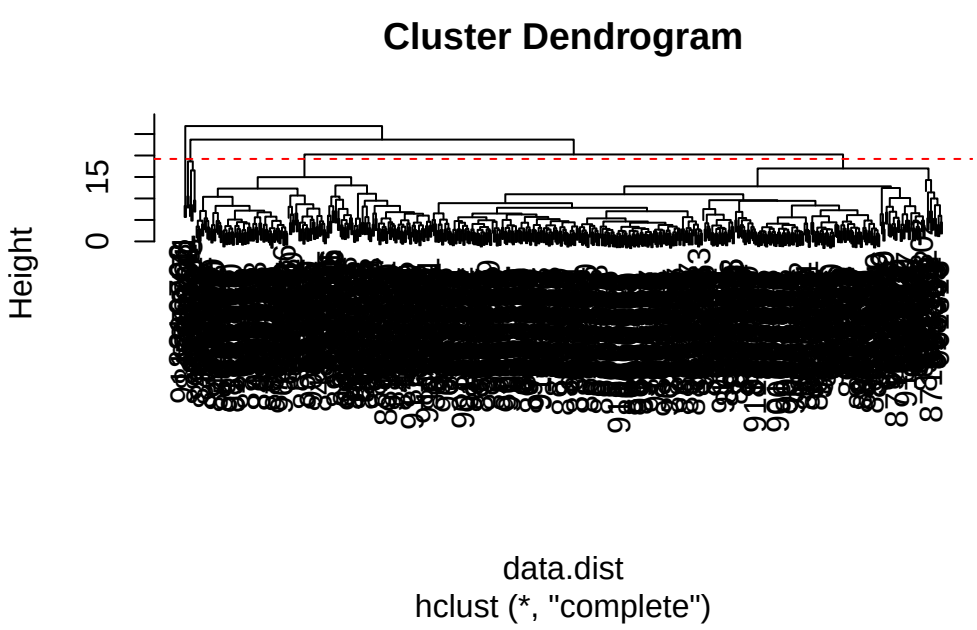
```
wisc.hclust <- ___(data.dist, ___)
```

Results of hierarchical clustering

Let's use the hierarchical clustering model you just created to determine a height (or distance between clusters) where a certain number of clusters exists.

- **Q10.** Using the `plot()` and `abline()` functions, what is the height at which the clustering model has 4 clusters?

```
plot(___)
abline(___, col="red", lty=2)
```



Selecting number of clusters

Normally in unsupervised learning we don't have labels to check our work against. Here we're lucky—the pathologist's diagnosis lets us evaluate whether our clusters correspond to something biologically meaningful. Let's see how well the clustering separates malignant from benign samples.

Use `cutree()` to cut the tree so that it has 4 clusters. Assign the output to the variable `wisc.hclust.clusters`.

```
wisc.hclust.clusters <- ___
```

We can use the `table()` function to compare the cluster membership to the actual diagnoses.

```
table(wisc.hclust.clusters, diagnosis)
```

```
                    diagnosis
wisc.hclust.clusters   B   M
                   1  12 165
                   2   2   5
                   3 343  40
                   4   0   2
```

Here we picked four clusters and see that cluster 1 largely corresponds to malignant cells (with `diagnosis` values of "M") whilst cluster 3 largely corresponds to benign cells (with `diagnosis` values of "B").

Before moving on, explore how different numbers of clusters affect the ability of the hierarchical clustering to separate the different diagnoses.

- **Q11.** OPTIONAL: Can you find a better cluster vs diagnoses match by cutting into a different number of clusters between 2 and 6? How do you judge the quality of your result in each case?

  Tip

  This not always an easy task and it really depends on how you define the "quality" of your results. We will go some way towards this in section 6 below.

Using different methods

As we discussed in our last class videos there are number of different *"methods"* we can use to combine points during the hierarchical clustering procedure. These include `"single"`, `"complete"`, `"average"` and (my favorite) `"ward.D2"`.

- **Q12.** Which method gives your favorite results for the same `data.dist` dataset? Explain your reasoning.

Side-note: The `method="ward.D2"`creates groups such that variance is minimized within clusters. This has the effect of looking for spherical clusters with the process starting with all points in individual clusters (bottom up) and then repeatedly merging a pair of clusters such that when merged there is a minimum increase in total within-cluster variance. This process continues until a single group including all points (the top of the tree) is defined.

One of the problems with Cluster Analysis is that different methods may produce different results – There is generally no universally accepted "best" method. The good news is that if your data really has clear groups all methods will likely find them and give you similar results. However, in more challenging cases like this one it is best to try multiple algorithms and see what groups logically make sense. A common approach is to use a smaller dummy dataset with pre-determined groups that you can use to see which algorithm best recreates what you expect.

5. Combining methods

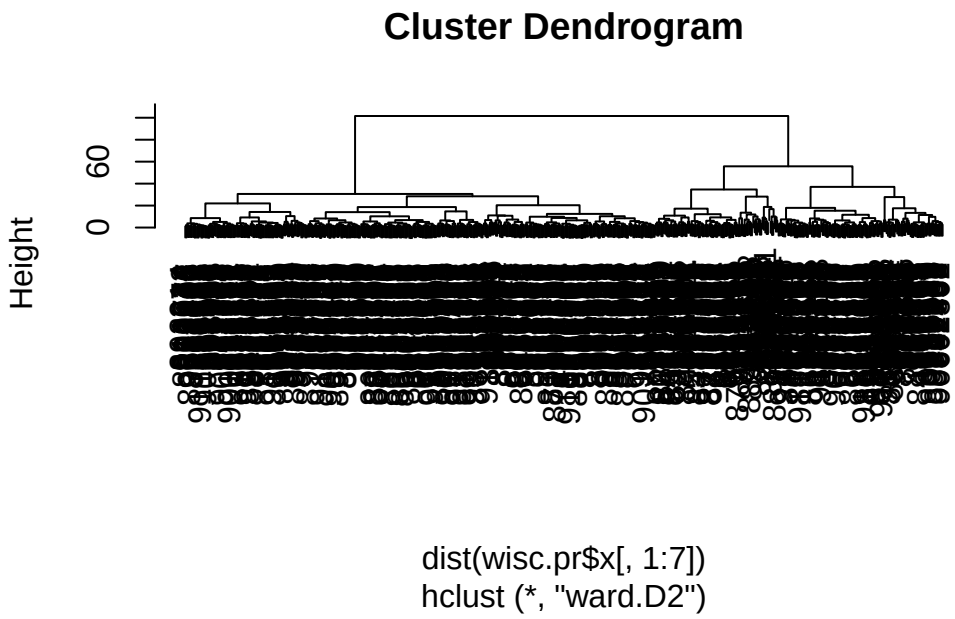
Clustering on PCA results

So far we have tried PCA and hierarchical clustering separately. Now let's combine them: cluster on the PC scores instead of the original 30 features.

Why might this help? PCA focuses on the directions of greatest variance—potentially giving the clustering algorithm a cleaner signal to work with.

Let's see if PCA improves or degrades the performance of hierarchical clustering.

Using the minimum number of principal components required to describe at least 90% of the variability in the data, create a hierarchical clustering model with the linkage `method="ward.D2"`. We use Ward's criterion here because it is based on multidimensional variance like principal components analysis. Assign the results to `wisc.pr.hclust`.



This looks much more promising than our previous clustering results on the original scaled data. Note the two main branches of our dendrogram indicating two main clusters - maybe these are malignant and benign. Let's find out!

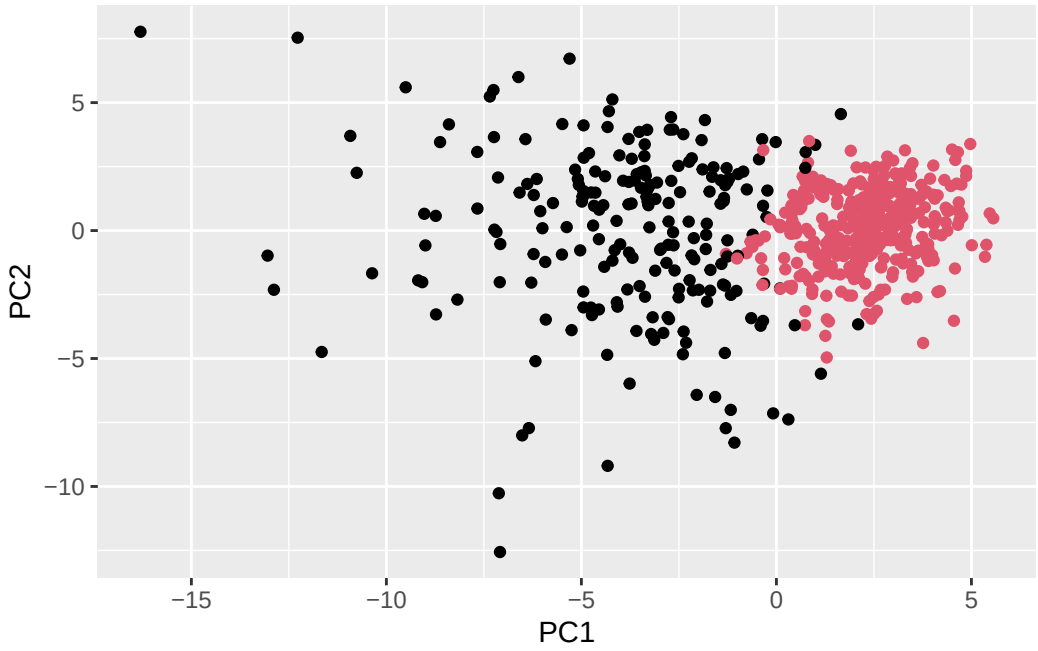
```
grps <- cutree(wisc.pr.hclust, k=2)
table(grps)
```

```
grps
  1   2
216 353
```

```
table(grps, diagnosis)
```

```
    diagnosis
grps   B   M
   1  28 188
   2 329  24
```

```
ggplot(wisc.pr$x) +
  aes(PC1, PC2) +
  geom_point(col=grps)
```




```
## Use the distance along the first 7 PCs for clustering i.e. wisc.pr$x[, 1:7]
wisc.pr.hclust <- hclust(___, method="ward.D2")
```

Cut this hierarchical clustering model into 2 clusters and assign the results to `wisc.pr.hclust.clusters`.

```
wisc.pr.hclust.clusters <- cutree(wisc.pr.hclust, k=2)
```

Using `table()`, compare the results from your new hierarchical clustering model with the actual diagnoses.

- **Q13**. How well does the newly created `hclust` model with two clusters separate out the two "M" and "B" diagnoses?

```
# Compare to actual diagnoses
table(___, diagnosis)
```

```
                      diagnosis
wisc.pr.hclust.clusters   B   M
                      1  28 188
                      2 329  24
```

- **Q14**. How well do the hierarchical clustering models you created in the previous sections (i.e. without first doing PCA) do in terms of separating the diagnoses? Again, use the `table()` function to compare the output of each model (`wisc.hclust.clusters` and `wisc.pr.hclust.clusters`) with the vector containing the actual diagnoses.

```
table(___, diagnosis)
```

```
                    diagnosis
wisc.hclust.clusters   B   M
                   1  12 165
                   2   2   5
                   3 343  40
                   4   0   2
```

6. Sensitivity/Specificity

Sensitivity refers to a test's ability to correctly detect ill patients who do have the condition. In our example here the sensitivity is the total number of samples in the cluster identified as predominantly malignant (cancerous) divided by the total number of known malignant samples. In other words: TP/(TP+FN).

Specificity relates to a test's ability to correctly reject healthy patients without a condition. In our example specificity is the proportion of benign (not cancerous) samples in the

cluster identified as predominantly benign that are known to be benign. In other words: TN/(TN+FP).

- **Q15.** OPTIONAL: Which of your analysis procedures resulted in a clustering model with the best specificity? How about sensitivity?

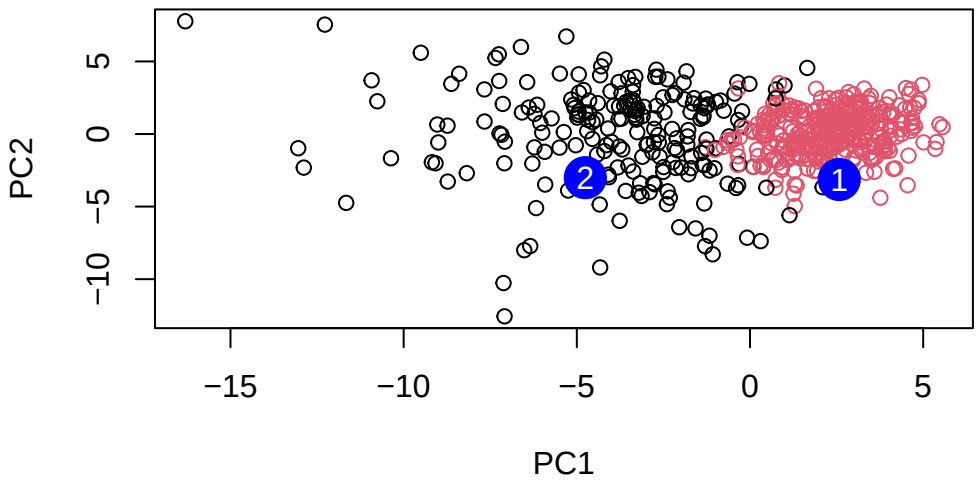
7. Prediction

We will use the `predict()` function that will take our PCA model from before and new cancer cell data and project that data onto our PCA space.

```
#url <- "new_samples.csv"
url <- "https://tinyurl.com/new-samples-CSV"
new <- read.csv(url)
npc <- predict(wisc.pr, newdata=new)
npc
```

```
          PC1       PC2        PC3        PC4       PC5        PC6        PC7
[1,]  2.576616 -3.135913  1.3990492 -0.7631950  2.781648 -0.8150185 -0.3959098
[2,] -4.754928 -3.009033 -0.1660946 -0.6052952 -1.140698 -1.2189945  0.8193031
           PC8       PC9       PC10      PC11      PC12      PC13     PC14
[1,] -0.2307350 0.1029569 -0.9272861 0.3411457  0.375921 0.1610764 1.187882
[2,] -0.3307423 0.5281896 -0.4855301 0.7173233 -1.185917 0.5893856 0.303029
          PC15       PC16        PC17        PC18        PC19       PC20
[1,] 0.3216974 -0.1743616 -0.07875393 -0.11207028 -0.08802955 -0.2495216
[2,] 0.1299153  0.1448061 -0.40509706  0.06565549  0.25591230 -0.4289500
          PC21       PC22       PC23       PC24        PC25         PC26
[1,]  0.1228233 0.09358453 0.08347651  0.1223396  0.02124121  0.078884581
[2,] -0.1224776 0.01732146 0.06316631 -0.2338618 -0.20755948 -0.009833238
             PC27        PC28         PC29         PC30
[1,]  0.220199544 -0.02946023 -0.015620933  0.005269029
[2,] -0.001134152  0.09638361  0.002795349 -0.019015820
```

```
plot(wisc.pr$x[,1:2], col=grps)
points(npc[,1], npc[,2], col="blue", pch=16, cex=3)
text(npc[,1], npc[,2], c(1,2), col="white")
```



- **Q16.** Which of these new patients should we prioritize for follow up based on your results?

8. Summary

In this mini-project, you applied unsupervised learning techniques to real-world cancer cell data from fine needle aspiration biopsies. You learned how to:

- Apply PCA to reduce dimensionality while retaining meaningful variance
- Interpret PCA results through scree plots, score plots and loading vectors (the 3 main result figures from PCA)
- Perform hierarchical clustering with different linkage methods
- Combine PCA with clustering for improved separation of sample groups
- Evaluate clustering performance using sensitivity and specificity metrics
- Use your trained PCA model to make predictions on new patient samples

A key takeaway is that preprocessing with PCA before clustering often yields better results than clustering on raw data alone. This workflow (dimensionality reduction followed by clustering) is widely used in biomedical research for tasks like patient stratification, biomarker discovery, and disease subtyping.

About this document

Here we use the `sessionInfo()` function to report on our R systems setup at the time of document execution.

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.7.3

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  l

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] rgl_1.3.18       factoextra_1.0.7 ggplot2_4.0.1    labsheet_0.1.2

loaded via a namespace (and not attached):
 [1] generics_0.1.3     tidyr_1.3.1        rstatix_0.7.2      digest_0.6.37
 [5] magrittr_2.0.3     evaluate_1.0.3     grid_4.4.2         RColorBrewer_1.1-
3
 [9] fastmap_1.2.0      jsonlite_2.0.0     processx_3.8.6     ggrepel_0.9.6
[13] backports_1.5.0    chromote_0.5.0     Formula_1.2-5      ps_1.9.1
[17] promises_1.3.2     purrr_1.0.4        scales_1.4.0       abind_1.4-8
[21] cli_3.6.4          rlang_1.1.6        base64enc_0.1-3    withr_3.0.2
[25] yaml_2.3.10        tools_4.4.2        ggsignif_0.6.4     dplyr_1.1.4
[29] ggpubr_0.6.0       broom_1.0.8        png_0.1-8          vctrs_0.6.5
[33] R6_2.6.1           lifecycle_1.0.4    car_3.1-3          htmlwidgets_1.6.4
[37] pkgconfig_2.0.3    pillar_1.10.2      later_1.4.2        gtable_0.3.6
[41] glue_1.8.0         Rcpp_1.0.14        xfun_0.52          tibble_3.2.1
[45] tidyselect_1.2.1   rstudioapi_0.17.1  knitr_1.50         farver_2.1.2
[49] websocket_1.4.4    htmltools_0.5.8.1  rmarkdown_2.29     carData_3.0-5
```

```
[53] labeling_0.4.3     webshot2_0.1.1     compiler_4.4.2     S7_0.2.0
```